MOSS Proof of Concept:
Validation Testbed Concept of Operations and System Under Test Requirements

draft: 2007-11-14

This document (informally called "the conops") describes the concept of operations of the NIST-developed validation tool (*i.e.* the "testbed") supporting the AIAG MOSS project, and the requirements of a system under test in its interaction with the testbed.

1 Goals of the Proof of Concept

The MOSS Proof of Concept (PoC) is a validation exercise intended to (1) validate the consistency of the MOSS recommendation itself, and (2) assess the ability of a technology provider's system to implement message formats conforming to the recommended practice. The MOSS Validation Testbed is a suite of software tools developed and managed by NIST and other MOSS participants for the purpose of performing these key tasks.

With regards to (1) validating the consistency of the MOSS recommendation itself, the recommendation identifies improvements in process and message specification that should be validated in the PoC exercise. Specifically:

- MOSS identified information elements essential to moving the goods.
  Are we correct? Is there more?
- MOSS recommends process modifications, to improve visibility, communication and collaboration. *Are these reasonable?*
- MOSS defined a conceptual model that identifies key concepts of the supply chain domain and their properties.
  Is our model correct? Sufficient?
- MOSS recommends use of UN eDocs and defines a "subset" for automotive.
  Is it sufficient?

With regard to (2) assessing the ability of tools to implement the message formats, the testbed tooling implements a "testing harness" that can be used by a "system under test" to assess its conformance to the message specifications. The details of how this is performed and what requirements it places on the interaction are the subject of this report.

1.1 Additional Notes

(a) The MOSS recommendation is not complete at the time of this writing. The technical content required for implementation is nearly complete, but there is currently no single definitive document. It is likely that we (testbed developers and supply chain software providers) will use a pre-normative mapping of information to message type structures in the course of the PoC, and finalize the document for publication

after the PoC or perhaps after the Pilot exercise.

(b) NIST participants in the MOSS project believe that the testbed that it is developing may provide value to NIST beyond its role in the MOSS PoC. The most obvious use beyond the PoC is in NIST's subsequent participation in the MOSS Pilot. There may be yet further, unforeseen value in the development of the tool. For these reasons, some of the presentation of the concept of operations of the testbed may seem unduly complex and indirect. We will try to be clear about the roll-out of capabilities, and what portion of it we believe is of immediate interest of PoC participants, but the complexity is inevitable if this document is to serve both the NIST developers and the PoC participants.

2 Use of the Testbed

2.1 Scope of Testing

The purpose of the PoC and Pilot exercises collectively are to validate the MOSS recommendation as described in *Section 1*. For practical reasons, these various types of validation cannot all be achieved simultaneously. The Testbed is focused on evaluating the ability of software tools to implement message formats conforming to the recommended practice. In the process of making this evaluation, we may discover problems in the MOSS specification including information elements left unspecified yet critical to the business process, details that should have been specified, and other concerns relevant to wider aspects of validation. These discoveries, while very important, are not by design the direct consequence of using the Testbed.

Capabilities of the Testbed will be delivered as they are implemented, and with the initial focus on what can be achieved through examination of messages emitted by the system under test. The capabilities planned for the initial December, 2007 "first run" of the Testbed are restricted to checking the correspondence of information elements across messages (*e.g.* that an item description present in an invoice message of a shipment is preserved in the bill of lading). Several MOSS participants have expressed the need to validate behaviors well beyond this simple test (*e.g.* changes to order quantity, change of planned transportation mode in the shipment's on-carriage, and reporting of various exceptional conditions). Though these are all within the scope of our future plans for the Testbed, they are not part of the December first run. Prioritization and planning for these tests with MOSS stakeholders must occur first.

Testing capability available in the December time frame is limited to *unit testing* where a single system under test is examined in the context of inputs (messages) that are provided by the testbed, and outputs (a message) are provided directly to the testbed for analysis. The focus of this unit testing is *conformance testing,* testing that assesses whether tool characteristics' violate normative statements of a specification.

Unit testing is contrasted with *system testing* where there might be multiple systems under test (the "units") exchanging messages among themselves. *Interoperability testing* is system testing that assesses the ability of tools to share information among each other, and thus the ability of parties possessing differing tools to work jointly toward a goal.

One rationale for limiting our work to conformance testing is that it simpler to implement and manage than an interoperability testing project. A second rationale is that conformance ought to lead to interoperability: Consensus standards projects such as MOSS seek to find agreement among the participants on the the structure of message types and the meaning of their information elements. These agreements lead software developers to a common interpretation of message content. When the

interpretation of the content is consistent, interoperability among tools is likely to follow.

Testing in the PoC is based on test scenario that describe abstract business processes. The abstraction elides low-level roles and tasks. Specifically the roles involved in the abstract business processes are those of MOSS Strata 1 and 2. (See *Figure 1*).

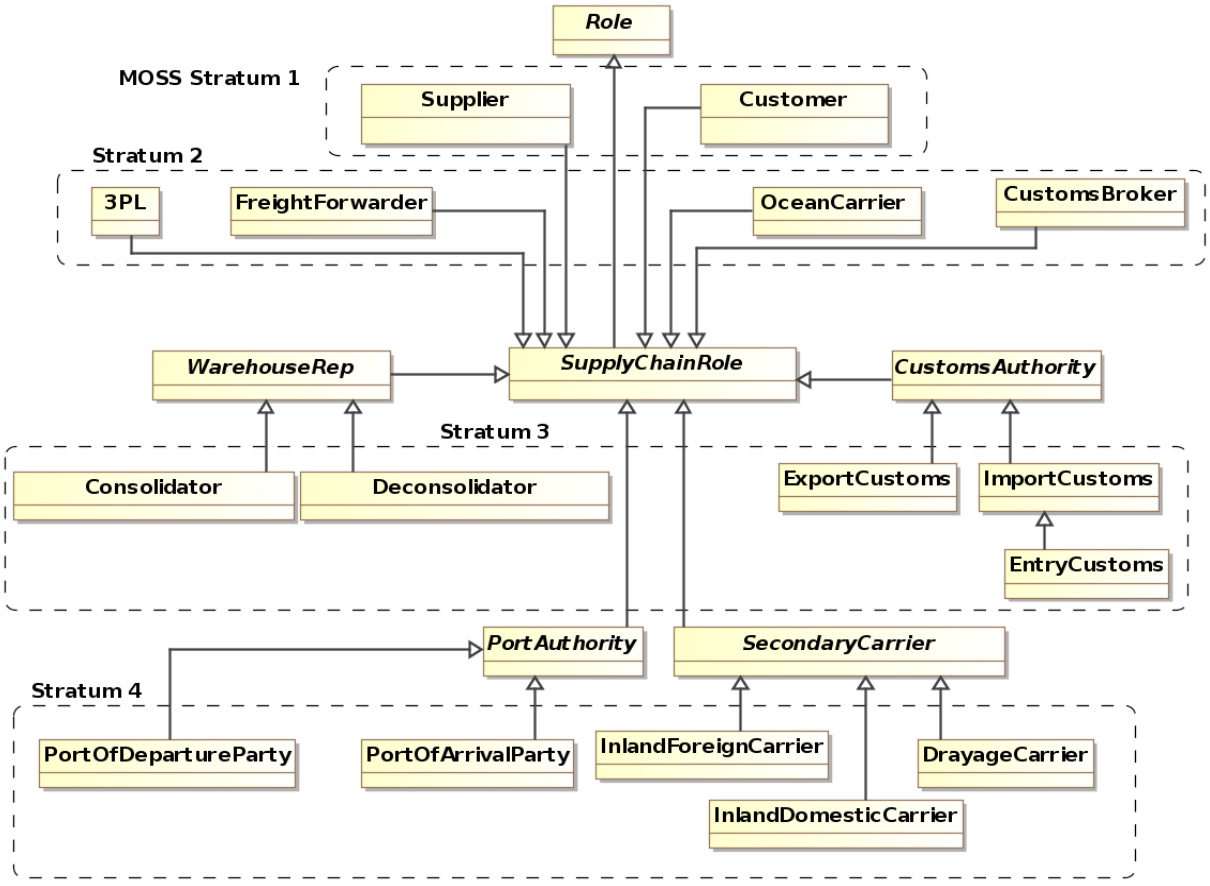


Figure 1: MOSS Strata

2.2 Test Scenarios and Exercises

Testing is performed in units described by test scenarios that describe a portion of a complete business process corresponding to some trade lane and some assignment of tasks to parties. *Figure 2* is a UML sequence diagram depicting a "door to door" process with the OceanCarrier providing all legs of the transport. The diagram elides detail of communication that might occur below the level of MOSS Strata 2 except that actors who are the recipient of a message from a Strata 2 actor are depicted. A test is performed as follows:

1. The system under test assumes the role in which it is commonly deployed (e.g. if it is a supplier's system, it assumes the Supplier role).
2. The system under test processes all messages directed to that role up to the process step that is being examined (e.g. If we are examining the Load Tender or EDIFACT INVOIC, the tool sees the EDIFACT DELJIT. See *Figure 2*.) The messages "directed to that role" are MOSS-conforming messages provided by the Testbed.

3. The system under test responds with the message designated by the test scenario as the output message (e.g. an INVOIC message).
4. The Testbed examines the message emitted by the system under test (the "output message" e.g. an INVOIC message) and provides a report on its conformance and completeness.

Figure 2 : Sequence diagram for door-to-door transport involving MOSS Strata 1 and 2.

The purpose of a test such as illustrated by the above is to assess the ability of the system under test to process the MOSS-conforming messages it would see in the portion of the process leading up to the generation of the output message, and to produce a MOSS-conforming message whose details are consistent with the information content of the messages it received. More specifically, tests of consistency may include ascertaining:

- Preservation of information elements that the MOSS recommendation has identified as referenced (not originated) in the output message. For example, the Customer's DUNS number from an ORDER message might need to appear in a particular field of the output message.
- Valid transformation of information elements as might be appropriate to reflect a change requested in the detailed of some received message (e.g. a request to reduce the quantity ordered, or cancel order of some item was received).
- Valid introduction of information elements identified by MOSS as originating with the output message. (The information to be introduced would typically be described in the narrative of the exercise).
- Adequacy of the information content of messages setting up the exercise, and that provided by the narrative, to enable the tool to produce the output message.

The message types that will be employed in the initial use of the testbed are the EDIFACT (version D.05B) messages identified in *Figure 2* specialized by means of the MOSS project's message implementation guildines (MIGs). At the time of this writing, the MOSS MIGs are in the form of spreadsheet (the "data element matrix") which identifies the data elements and the encoding of these to be used in each of the messages used in the scenarios. Where ANSI X12 message types appear in *Figure 2*, the data element matrix will align the X12 data element definitions with the UNTDED definitions used in the EDIFACT messages.

The choice of EDI over XML-based messages reflects the expressed interests of the participants, particularly the OEMs. The Testbed is being designed to support both XML- and EDI-based messaging.

An *exercise* is the unit of testing that involves the download (from the testbed) and processing of messages of some scenario's business process, and the generation, upload and analysis of a single *output message* (the system under test's response to the "problem" posed by the exercise). The messages downloaded are said to be *setting up the exercise*.

The first exercise planned, *Exercise 1*, involves the production of an EDIFACT INVOIC message. The (single) message setting up the exercise is an EDIFACT DELJIT message of a scenario that is the door-to-door shipment of goods from Europe to the US, where the pre-carriage, main carriage and on-carriage are all performed by the OceanCarrier. The sequence diagram associated with this exercise is provided by *Figure 2*.

Messages setting up the exercise are the only interface to the system under test assumed to exist. They may be the sole source of knowledge of the scenario, or it may be that the system under test has been configured by some means, unknown and unrelated to the testbed, to perform the exercise.

2.3 Detailed Requirements on Interaction with the Testbed

The Testbed software will interface with its users through an Apache HTTP server, located at NIST. The homepage to this service will be http://syseng.nist.gov/moss as is used now for other NIST-developed MOSS tools. In the December time frame, only a primitive, manually operated interface will be provided. In this arrangement, the messages setting up the exercise in the system under test will be available for download on a page dedicated to the exercise. The response provided by the system under test can be uploaded to the server using the usual HTTP file upload (RFC 1867) provided by web browsers. The server will respond to an upload with a report of its analysis of the emitted message (the file uploaded).

As this arrangement suggests, in the December time frame, testing is performed "off-line" where the

operators of the system under test are free to upload the output message at any time. There is no synchronization between the download of messages setting up the exercise and the upload. There will be one HTML page for each exercise, providing both the messages setting up the exercise and the upload form. The essential requirement on the users is to upload the output message at the same page that he downloaded the messages setting up the exercise, since otherwise the analysis performed will not be the one appropriate for the messages setting up the exercise that he downloaded.

3 Concept of Operations

This section provides a high-level view of the purpose, architecture and behavior of the NIST-developed MOSS Validation Testbed. The details here are perhaps only of interest to the NIST team developing the Testbed.

3.1 Requirements Addressed

The Testbed is being developed with the requirements of the December time frame PoC testing foremost. However, NIST envisages additional roles for the Testbed, and use, beyond that time frame. NIST intends to incorporate development of the Testbed within its more comprehensive research agenda. The requirements that the research agenda place on the design of the Testbed may have lead to a design that is more challenging to develop than if these requirements were not posited.

NIST intends to use this work to explore the following:

(1) The use of a domain conceptual model, defined as UML class diagrams, to describe objects and relationships in the domain of long-distance supply chains;

(2) The ability to automatically translate this UML model to axioms of a first-order logic domain ontology;

(3) The use of this domain ontology, extended with additional axioms, and employing a first-order logic reasoner, to identify inconsistencies among the information provided by the messages setting up the exercise and the output message, collectively;

(4) The use of a structural information mapping specification (QVT) to associate information elements of message structures with statements in the conceptual model of (1);

(5) The use of a structural information mapping engine to populate, from message content, a knowledge base conforming to the conceptual model of (1);

(6) The use of a message type meta-model to increase the commonality of expression of mapping statements in the mapping specification of (4);

(7) The use of the information mapping specification of (4) as a means to express the semantics of the message elements through their association to elements of the conceptual model of (1).

It would be excessively optimistic to assume that by December, 2007 any significant progress could be made on some of the items of this research agenda. Particularly, (7) will not be explored until much later, and (3) will be performed in a parallel effort enabled by providing the ground facts produced through the mapping (5) to first-order logic tooling that is not integrated with the Testbed. Instead of the planned fully-integrated first-order logic reasoning capability, the Testbed, in the December time frame, will use explicit Datalog and/or OCL expressions to perform testing of the output message. We already possess and are familiar with the Datalog and OCL tooling we intend to use.

3.2 Overview of the architecture

This section provides a description of the Testbed system architecture.

The principle components of the system, as depicted in *Figure 3*, are:

- A *System Under Test* : a supply chain technology provider's tool that is the subject of testing. This is, of course, not actually a component of the testbed, but here for expository purposes.
- *Messages Setting up the Exercise* : EDI- or XML-based messages that correspond to some scenario, and some testing purpose, and are, in the business process, typically conveyed to the system under test.
- *Output Message* : an EDI- or XML-based message that is emitted from the system under test and is the subject of the conformance analysis peformed by the testbed.
- *QVT Mapping Engine* : a structural information mapping engine that maps message content (messages setting up the exercise, and the output message) to the knowledge base.
- *Message Models* : MOF-based models of the message types, one for each message type of the messages setting up the exercise, and the output message. These are each source schema for the mapping of one message type by the mapping engine.
- *MOSS Conceptual Model* : A UML model, consisting of class diagrams that describes object types and relationships in long-distance supply chains. This model is the target schema of the mapping engine (in all uses) and is translated to first-order sentences to provide portions of the ontology used by the first-order reasoner (The reasoner is not depicted in *Figure 3*, but its function is depicted by the peach color box).
- *Trade Lane and Business Process Knowledge* : Additional axioms (additional to those generated from the UML model) that provide constraints and ground facts (e.g. that there is a port in Oakland, California) that are relevant to the scenario.
- *Knowlede base* : that is populated by the ground facts generated by mapping messages to a population conforming to the conceptual model (role of the mapping engine), and by ground facts provided by the Trade Lane and Business Process Knowledge.
- *MOSS Message to Model Mapping Specs* : QVT mapping specifications that map message content (conforming to the message model source schema) to a conceptual model population (conforming to the conceptual model, the mappings target schema).
- *Report* : a report generated by evaluation of the axioms, (or in the December time frame, the OCL constraints or Datalog queries). The report describes the assessment of conformance and completeness made of the system under test's ability to generate the output message.

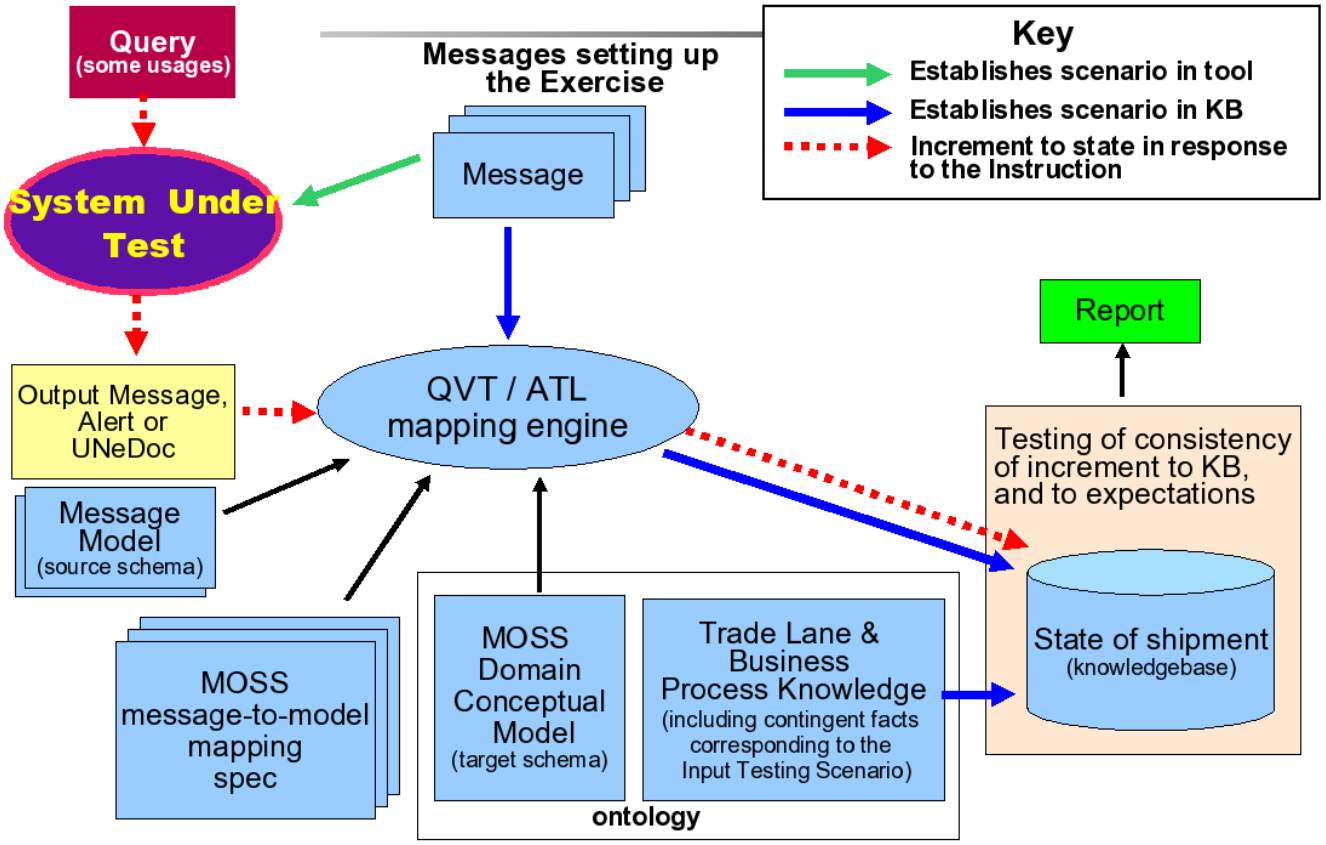


Figure 3: High-level architecture of the MOSS Validation Testbed (unit test view)

In the processing of an exercise, the system under test and the testbed itself are both provided with messages setting up the exercise. The system under test will use these to establish the context of the scenario in its internal representation. The testbed will similarly set up the context – it does this by translating the message content into statements that become part of the knowledge base (*State of shipment* in *Figure 3*). The means by which it performs this translation is a mapping performed by the QVT mapping engine. The source schema to this mapping is a message model corresponding to the message being processed (i.e. each messages setting up the exercise, in turn). The target schema of these mappings is the conceptual model. With this work performed, the tool under test may then generate the output message. Like the messages setting up the exercise, this message is processed through the mapping engine to statements that become part of the knowledge base. Unlike the content derived from the messages setting up the exercise, however, these statements may contradict statements made previously. Contradictions indicate non-conformance to the recommended practice, and are the subject of the Report generated. Omission of information that is mandatory in the message will also be noted in the mapping of the output message.

The output message may introduce content not found in the previous messages of the business process. For example, a load tender message may specify dimensions and weights of packages – this information is not found in the DELJIT message that initiated the shipment and set up the exercise. To enable testing of these items, the testbed must be made aware that new information is being introduce, and it must be provided with the details. This is accounted for as Trade Lane and Business Process Knowledge (And thus the verbiage *"including contingent facts corresponding to the Input Testing Scenario"* in *Figure 3*.). The test exercise will provide direction to the operators of the tool under test with regard to such issues. For example, the narrative of the exercise may include the statement "Assume that the goods are to be packaged into a box of dimensions 1m x .5m x 1.5m."

The guiding principle of the design of the testbed is that the system under test, like all software systems, is designed around some "implicit ontology," (in this case, of the supply chain transport domain) – it possesses and manipulates formalisms of its implicit ontology, and its behaviors (*e.g.* the generation of messages) is governed by the "rules" of that ontology. We, as developers of validation tools, cannot inspect the implicit ontology of the system under test. We can, however, describe a "parallel ontology" conforming to the MOSS conceptual model and MOSS recommended practices. And we can interpret output from the tool under test with respect to the parallel ontology. The process of interpreting the output message provides either sentences confirming what we know about the scenario (through our parallel, reference interpretation and ground facts of the trade lane and business process) or sentences contradicting what we know about the scenairo. In this latter case the tool in non-conforming with respect to the scenario.

The matter of interpreting the output message with respect to "a reference ontology" is, in fact, the key task of validating the tool under test with respect to the specification (e.g. the MOSS recommended practice). In as far as we can formalize the interpretation of message content into a population consistent with the domain ontology, we have pointed the way toward more formal, and possibly more effective means of expressing the message specifications and recommended practice.

3.3 Meta-model Architecture

Figure 3 elides detail of how information is moved from message content into structures based on the conceptual model. This task, and also the task of generating messages setting up the exercise, is performed through a meta-model architecture, based on OMG technologies. This section describes this tooling.

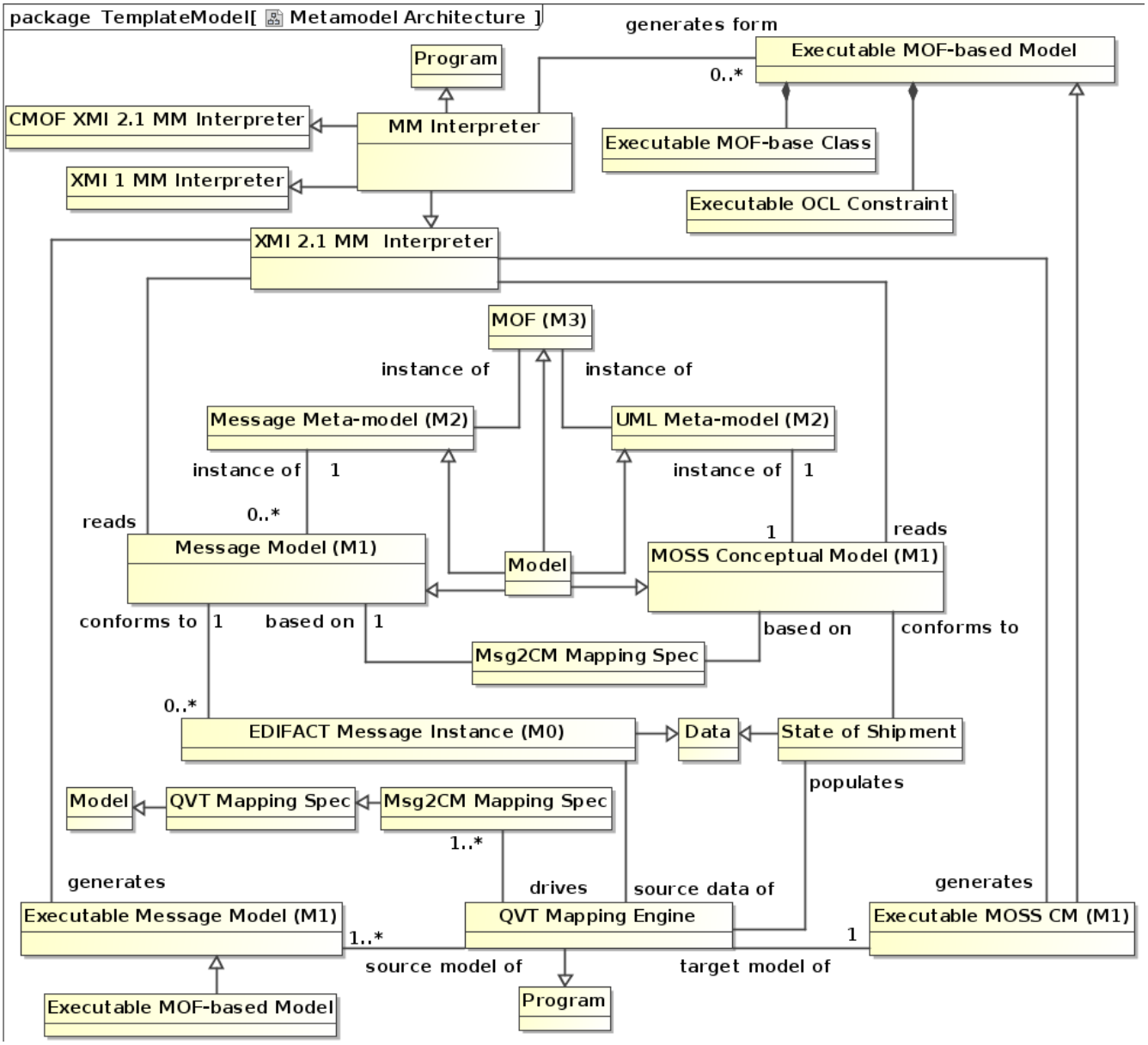


Figure 4 : The Meta-model Architecture.

Figure 4 describes components of the architecture.[1] *MM Interpreter* is a class of programs that produce executable MOF-based models from XMI input. For various projects that use this framework, we have developed instances of three sublcasses of MM Interpreter that are distinguished only by the form of input they accept: CMOF XMI 2.1, and UML Class diagrams in XMI 1.1 and XMI 2.1). In MOSS it is anticipated that we will only use the subtype *XMI 2.1 MM Interpreter*. It is used to produce an Executable MOF-based model of the message model M1 (*Executable Message Model (M1)*) and the executable version of the conceptual model, *Executable MOSS CM (M1)*.

Executable Message Models are generated from corresponding *Message Model (M1)* models. The Message Model (M1) is depicted in *Figure 5*. One *Message Model (M1)* is generated for each of the message types in the PoC (e.g. the EDIFACT DELJIT, the INVOIC, etc.) by a program that reads the MOSS Data Matrix Table.

1 Some of what is depicted as generalization might more accurately be described as instantiation, but relationships between instances and classes are not permitted in UML.

Mapping is performed by the *QVT Mapping Engine* using an *Executable Message Model* for one message as source model and the *Executable MOSS CM (M1)* as target model. For this combination of source and target models, a message-to-conceptual model mapping specification is defined (*Msg2CM Mapping Spec*). Execution of the mapping engine, with these specifications and an *EDIFACT Message Instance* conforming to its corresponding *Message Meta-model (M1),* populates the *State of Shipment*, a data population conforming to the *MOSS Conceptual Model (M1)*. Additional tooling validates that the population is well-formed. This tooling makes use of cardinality, subsetting, and type information defined in the *Executable MOF-based* class, and the *Executable OCL*, if any. The sort of validation performed by these means is only loosely related to the validation goals described in *Section 2.2.* Rather, what is performed here primarily serves to ensure that the mapping specification is valid.

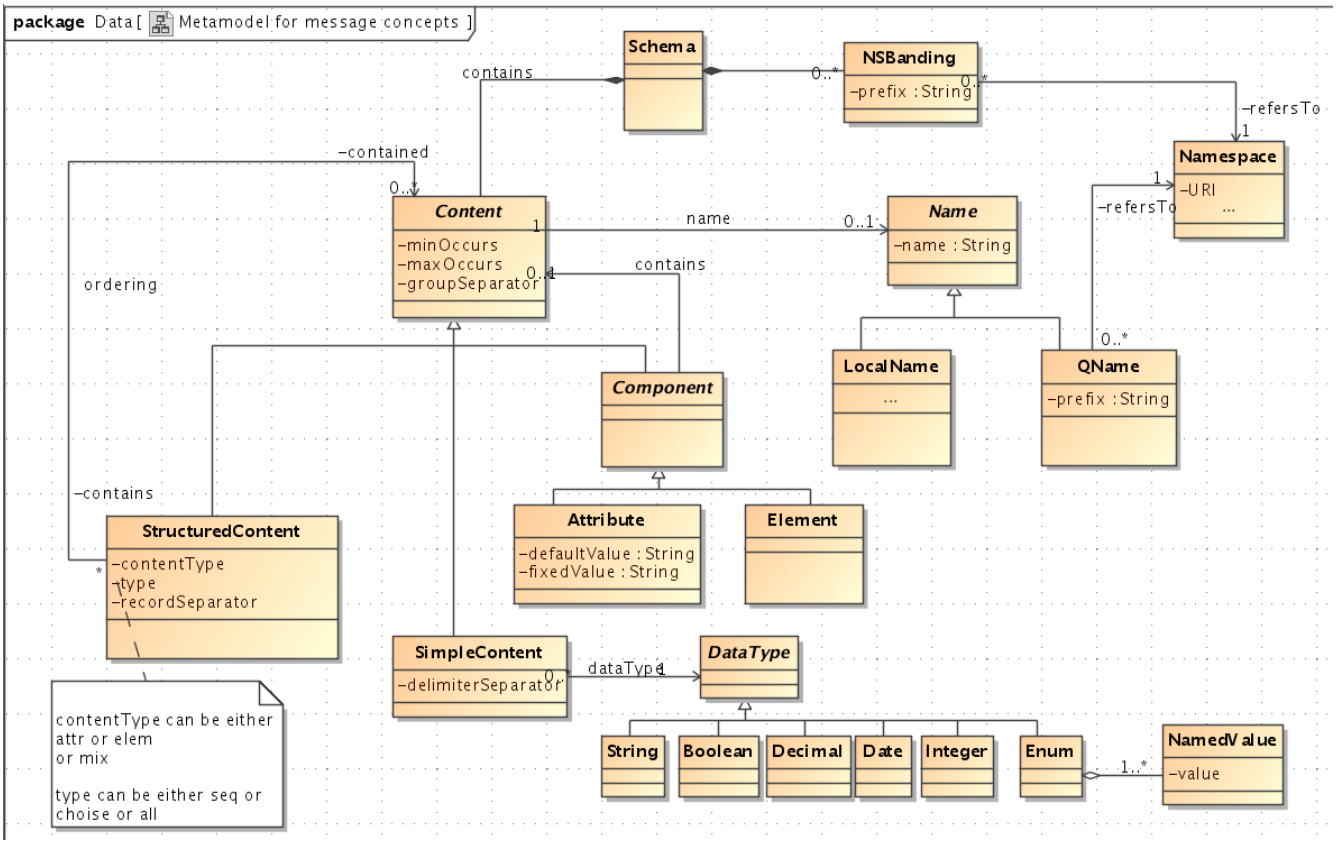
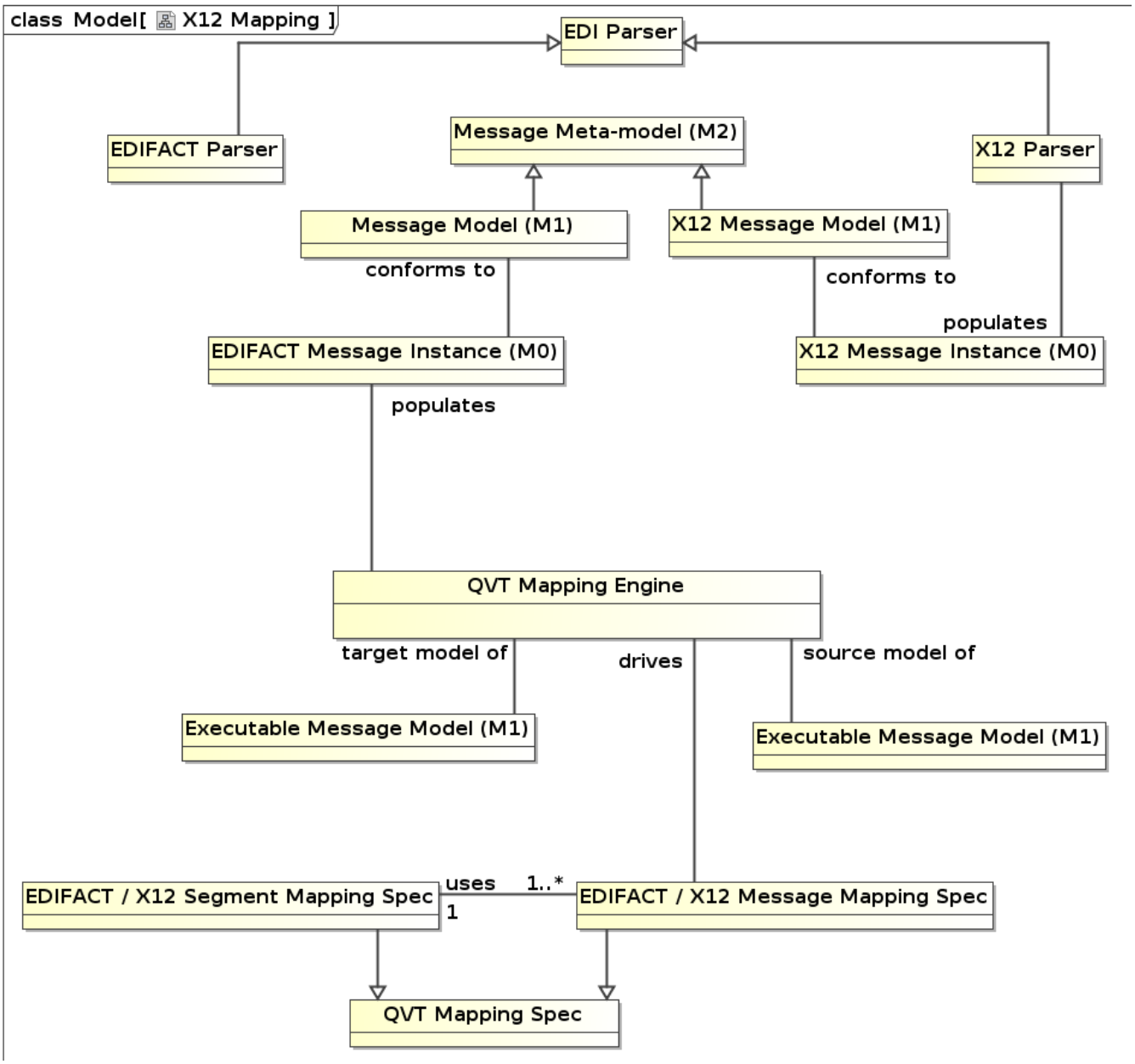


Figure 5: The meta-model of messages to which the various EDI meta-models conform.

3.3.1 The Architecture and ANSI X12 Support

The meta-model architecture and mapping engine technology that is used to support the translation of message content into conceptual model structures can also provide means to translate between X12 and EDIFACT message content. In this role, QVT mappings are specified between segment type definitions used in these two EDI frameworks (*EDIFACT / X12 Segment Mapping Spec*, in *Figure 6*, below). These mapping specifications, for each of the segment types used in the PoC (roughly 20) are reused by each of the *EDIFACT / X12 Mapping Spec*s, one of these for each message type used in the PoC for which an X12 or EDIFACT equivalent is needed. In the mapping of the X12 204 (a load tender message type), for exmple, an 'anonymous' (not corresponding to a standard message type) equivalent EDIFACT message instance (*EDIFACT Message Instance (M0)*) could be created. This data could then be used in the

translation of message content into conceptual model structures, with traceable relation back to the original data.



4 Glossary

conformance testing – testing that assesses whether tool characteristics' violate normative statements of the specification

interoperability testing – testing that assesses the ability of tools to share information among each other, and thus the ability of parties possessing differing tools to work jointly toward a goal.

MOSS recommended practice – a document describing the findings of the MOSS project, including recommendations for an improved business process, message formats, and message mapping to a conceptual model. *N.B.* : At the time of this writing, this document does not exist. However, most of what it will recommend has been identified. The Testbed is being developed with reference to the intermediate work products of the project.

output message – the message emitted by the system under test, for a given exercise, that will be examined by the testbed for conformance to the MOSS recommended practice

system under test – a software tool that intends to support the MOSS recommended practice by processing and emitting messages conforming to the recommendation, and is interacting with the testbed for the purpose of assessing its ability to process and emit these messages

validation – the systems engineering task of ensuring that a system meets stakeholder requirements

verification – the systems engineering task of ensuring that the methods applied in the development of the system are sound, appropriate to the problem at hand, and in fact, being applied